COTS-based OO-Component Approach for Software Inter-operability and Reuse (Software Systems Engineering Methodology)[1]

Laverne Hall & ReUse/OO-Component Team
Jet Propulsion Laboratory / California Institute of Technology
4800 Oak Grove Drive
Pasadena, CA 91109-8099, USA
Phone: 818 393-5430
Laverne.Hall@jpl.nasa.gov

Abstract - The purpose of this research and study paper is to provide a summary description and results of rapid development accomplishments at NASA/JPL in the area of advanced distributed computing technology using a Commercial-Off-The-Shelf(COTS)-based object-oriented component approach to open inter-operable software systems development and software reuse (i.e., COTS-based software components in action).

Distributed COTS middleware (such as CORBA ACE-TAO) coupled with well-defined layered software architecture can be used to support infrastructure development for object-oriented component technology. It can provide a framework for component development, legacy incorporation, and reuse and inter-operability across subsystems. With detailed systems engineering, it can reduce development, testing, and maintenance relative to life-cycle cost and time.

This paper will 1) address what is meant by the terminology object-oriented (OO) component software and how object component technology can be used in scientific software development and operational environments, 2) give an overview of the component-based implementation strategy and how it relates to infrastructure support of software architectures promoting reuse/inter-operability, and 3) evaluate the benefits or lessons learned from this approach (such as complexity of integration and avoiding duplication or re-development efforts).

TABLE OF CONTENTS

1. INTRODUCTION & BACKGROUND

Distributed computing allows modern software structure to occur across distributed networks in an increasingly flexible and effective manner. Software component technology allows distributed application pieces to flexibly be pluggable, reused, inter-operate, and evolve over time. Figure 1 shows the evolution of network infrastructure technology support for distributed computing and how the trend has gone from proprietary methods to continuous improvements using open standards. The focus here is to extend beyond taking advantage of object-oriented techniques to a more advanced open methodology for developing and executing software systems using the components concept (some areas still being researched and developed for standardization).

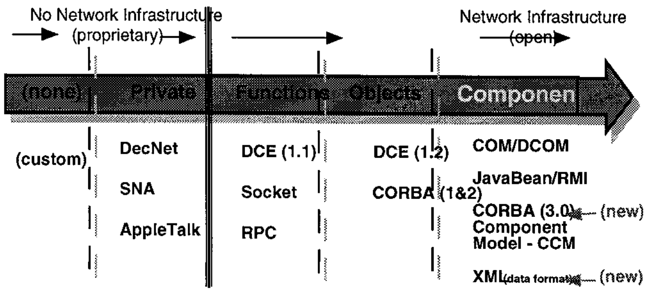


Figure 1 - Distributed Computing Technology Evolution

[1] 0-7803-6599-2/01/$10.00 © 2001 IEEE

The ReUse/OO-Component Team focused on providing advanced research solutions to software architectures via reusable infrastructure development and prototypes using new technology and using these solutions to mitigate risk during future task insertion. Many solutions are centered around the use of an advanced object-oriented distributed systems approach, currently object-oriented component-based software in particular, with actual code, templates, supporting framework elements, and sample application prototypes provided.

Two separate, but related component studies were conducted by the ReUse/OO-Component Team based on 2 different open specification models and were as follows:

1) Microsoft's Component Object Model (COM)
   - Refer to Reference [2] for details.
   - Used only interface specification or structuring technique for packaging components, done in a C++ and UNIX development environment.
   - Took JPL developed proprietary application code based on a Distributed Computing Environment (DCE) infrastructure and repackaged it into components with generic Application Program Interfaces (API) to hide the communication layer from developers or user subsystems.
   - The goal was to develop a tiered architecture centered around a component approach to allow for possible swapping of the underlying communication layer (and maybe other services) with other technology while keeping the same API's.
2) COTS-based CORBA model
   - ** Focus study of this paper.
   - Took same application used in the COM study, but replaced both propriety development and DCE with CORBA-based infrastructure service components while maintaining a easy mapping to the same API's used by existing subsystems, now hiding CORBA from the service users.

A more detailed scope of areas to be covered while discussing the above mentioned second model study include:

• Stating the motivation for and benefits of COTS-based component approach to software systems development, along with the objectives of the prototyped demonstration.

• Briefly describing the what is meant by an object-oriented component approach to development and giving examples of scientific applications which can take advantage of the components approach.

• Addressing the key support environment elements desired to make up a comprehensive component deployment and management strategy when developing systems using application configurable components.

• Providing a brief description of Common Object Request Broker Architecture (CORBA) standard and how it is being used in this study.

• Discussing a completed and demonstrated application prototype of portions of the NASA Deep Space Network (DSN) Monitor & Control Infrastructure Service (MCIS) using ACE-TAO (a COTS implementation of CORBA features).

• Discussing lessons learned from using CORBA ACE-TAO (pros and cons) as the underlying communications infrastructure and service provider to the applications.

2. MOTIVATION & OBJECTIVES

The purpose behind this COTS-based OO-component approach was to focus on a faster way for developing reusable common software services and supporting infrastructure which would improve on software engineering methodology by creating a framework or roadmap with greater, more flexible benefits for most distributed system development and their life cycle. Rapid development or early prototyping using this approach with existing applications give an opportunity for proof-of-concept and risk mitigation when using new technology for future application developments or upgrades. This is done by taking advantage of object-oriented techniques such as software design pattern frameworks, code wrapping, class inheritance/encapsulation, etc. in addition to coupling object-oriented techniques with software component structuring concepts to form reusable, configurable common services based on open standards. The OO-component approach provides for maximum use of proven COTS for added flexibility in product selection, upgrades, and complete or partial switch-overs. There is no inter-mingling of the COTS product inside the application-specific code.

There are several key areas or standards were addressed dealing with advanced technology capabilities; however, only certain ones were demonstrated in this effort. The following is a list of research techniques/technologies along with identification (*) of the one used in the prototype addressed in the results of this paper:

• OO (C++) code wrapping techniques and design patterns for reusable API's (*)

• distributed COTS-based services [CORBA (*), Java/RMI, COM/DCOM]

• CORBA Component Model (CCM), packaging structure for s/w services (similar to open spec on Component Object Model - MS/COM)

• Extensible Markup Language (XML) for universal data exchange (considered as future supporting option)

• Unified Modeling Language (UML) for design process and documentation (*).

There are many traditional problems encountered in the development and maintenance or upgrade of software systems which contribute to the motivation or need for a new approach to engineering these systems. For example, within the NASA's DSN, at least 60% of its subsystems have common software functions (often different versions or implementations of the same functions) and a lack of common infrastructure to support objectives for improved development, maintenance and inter-operability. Some typical examples of problems are as follows:

• too much customization and lack of straight-forward reusability - end up redoing or modifying
• complex system assembly and delivery
• lack of standards leading to better coordination of development systems and phases for complex applications
• lack of inter-operability among heterogeneous and scaleable systems or system components
• difficulty in coping with change which affects re-test and redelivery of many affected legacy subsystems
• need to reduce development and maintenance cycle time and cost (faster/better/cheaper, f/b/c)
• need to integrate legacy, COTS, or other technologies over time
• lack of runtime flexibility
• etc.

3. OO-COMPONENT APPROACH & APPLICATIONS

The component software approach promotes more advanced object-oriented component-based techniques and architectures into software development. It provides a standard framework or infrastructure for building and using software components to save time and money.

The top portion of Figure 2 gives a pictorial view of the goal to move from a traditional monolithic type application composed of a single binary file to applications which can be easily configured (static or dynamic, local or remote) from a library of components providing common services across applications. Applications would reduce their focus to developing custom or application-specific components to plug-and-play with generic or common services components.

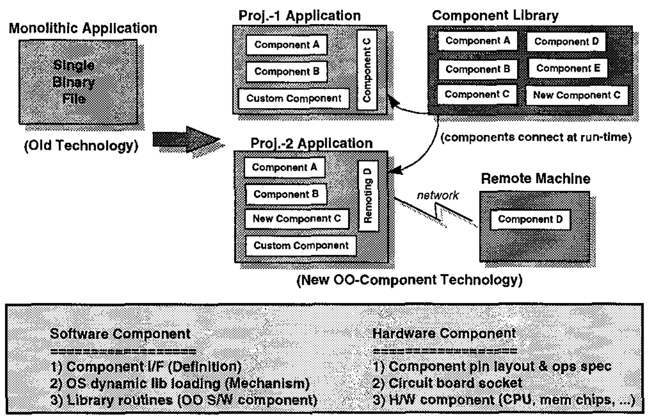


Figure 2 - OO-Component Architecture Concept

The objective is to design application software from a set of functional software components. Each software component has a well-defined interface that encapsulates a distinct operation. This is a similar process as to constructing a hardware circuit board with the hardware architecture equating to the elements of a software architecture using components (see bottom portion of Figure 2).

A summary of key benefits to using the OO-component software approach, which should lead to long-term cost and time savings through-out various life-cycle phases (up-front initial costs may be higher than ...), include the following:
1) inter-operability
2) extensibility
3) easy reuse
4) easy assembly
5) run-time flexibility
6) enforce design standards
7) development flexibility.

A COTS-based OO-component approach can allow application configurations to be produced quickly and can result in higher quality, more reliable software (** if the COTS capabilities are well understood and integratable).

This approach also provides for prototyped reusable generic object-oriented software components for common services, both communications and application services. Figure 3 shows examples of scientific applications, using a tiered architecture or methodology, which can take advantage of the component approach. A few application-specific examples include monitor and control (focus of this study and currently demonstrated), spacecraft rules subsystem for planning, and spacecraft modeling (future projections). The middle tier of the architecture would allow development of enabling support components (such as expression evaluators or user defined functions) and the lower layer would provide communication service components (which can be based on simple protocols like TCP/IP to complex ones like CORBA) for the applications.

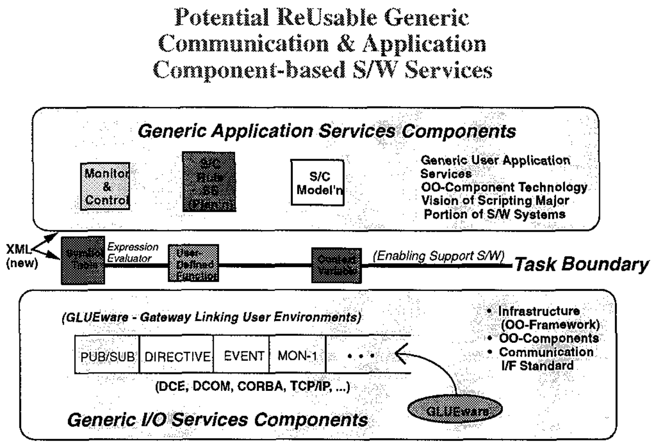

Potential ReUsable Generic Communication & Application Component-based S/W Services
Generic Application Services Components
Monitor & Control
S/C Model'n
Generic User Application Services
OO-Component Technology
Vision of Scripting Major Portion of S/W Systems
XML (new)
Expression Evaluator
(Enabling Support S/W)
Task Boundary
(GLUEware - Gateway Linking User Environments)
PUB/SUB
DIRECTIVE
EVENT
MON-1
• • •
(DCE, DCOM, CORBA, TCP/IP, ...)
• Infrastructure (OO-Framework)
• OO-Components
• Communication I/F Standard
GLUEware
Generic I/O Services Components

Figure 3 - Component-based Applications using Tiered Architecture

Utilization of the object-oriented component technology approach (with or without COTS) for system development and software reuse will apply to several areas within JPL, and possibly across other NASA Centers, for example:

• NASA/JPL Telecommunications & Mission Operations Directorate (TMOD) Deep Space Network (DSN)
• NASA/JPL Flight Software - Ex., Mission Data Systems (MDS)
• Other technology and applications programs - Ex., NASA/JPL Technology & Applications Programs (TAP) such as the Intelligent Synthesis Environment (ISE) for Distributed Modeling, Department of Defense (DOD) programs such as Defense Information Systems Agency (DISA) Defense Information Infrastructure (DII), etc.

4. COMPONENT DEPLOYMENT & MANAGEMENT STRATEGY

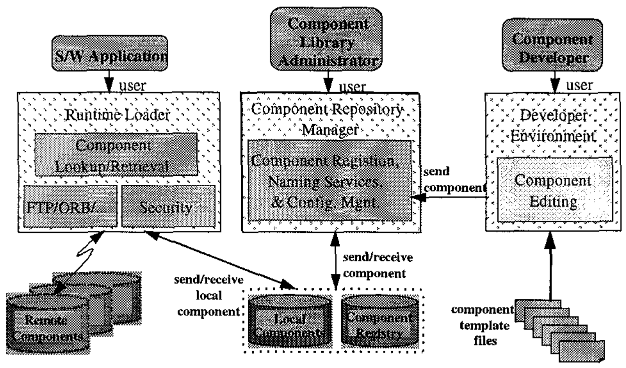
Component-Based Development (CBD) is the new paradigm for designing modular applications with reusable software entities. An example of components are COM components, Enterprise Java Beans, and to some degree CORBA Objects *(see CORBA Component Model).* CBD offers the promise of reducing cycle time and improving the quality of delivered applications, by building applications with modules with a well defined interface, providing a well defined service.

The goal of a Component Deployment & Management Strategy (CDMS) is to specify a design for a repository of components so large scale systems can be constructed. This is very good for standardization and will help solve the problem of continuous reinvention (i.e. just download the required component).

In order to implement CBD effectively, two major challenges must be solved:
1) A solution for storing the components in an accessible repository
2) Deploying the components to local or remote sites (developers, applications etc.)

The required support environment and the procedures for developing new software components and component deployment are addressed via the Object Component Deployment and Management Strategy (CDMS). Issues related to developing components, accessing the components repository or database during development and/or execution, component security, and needs for a component-capable configuration management (CM) system verses traditional are taken into account.

Figure 4 depicts the key elements needed to support the development of a distributed component-based system (based on both studies). CORBA was found to have extensive specifications providing most of these key elements (or support services); thus, reinforcing the use of an open standard COTS to avoid custom or proprietary efforts.



* Elements Recommended to Support Component Implementation Approach
* Build upon best elements of different standards to create a robust, generic framework

Figure 4 - Component Deployment & Management Strategy or Roadmap

Some of the remaining key challenges of a Component Deployment & Management Strategy are as follows:
• Component/Object versioning is still a research topic and is not fully resolved.
- *i.e. How do you version a component when the dependencies change?*
• The CORBA Component Model (CCM) addresses many of the CDMS issues (such as deployment, packaging, etc.), but is not yet fully implemented.
• Integration among different component models like EJB, COM, CCM is not yet well defined.
• Institutional process and implementation is required for this to succeed.

5. DISTRIBUTED ARCHITECTURE & COTS IMPLEMENTATION

Among the common distributed object architecture standards for middleware computing technologies are Common Object Request Broker Architecture (CORBA), Distributed Computing Environment (DCE), Java/RMI or Enterprise Java Beans (EJB), and Microsoft's Component Object Model (COM). This study focused on the use of CORBA.

CORBA is an open distributed object computing infrastructure standardized by the Object Management Group (OMG). CORBA is increasingly becoming the way the "rest of the world" creates distributed systems. It is simply not practical for large corporations to attempt to design and implement a distributed system infrastructure from scratch when already existing standards exist and have been in use for a long time.

Primary components in the CORBA architecture include (see Figure 5 architecture reference model):
• **ORB** (Object Request Broker)
- provides the basic communication channel through which objects interact to provide system services

• Object Services

- globally available fundamental services, e.g. Naming Service, Event Service, and Notification Service

• Common Facilities

- higher level services (e.g. User Interface) and domain-specific tasks (e.g. Distributed Simulation).

As depicted by Figure 5 and discussed below, CORBA services provide similar functions as those developed in-house by JPL (but prior to this study effort) for the same prototyped application discussed in the next section.

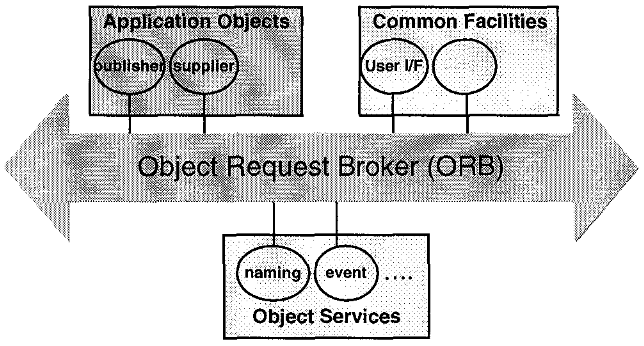


Figure 5 - CORBA (OMG Ref. Model Archit.)

The COTS implementation of CORBA specifications used in this particular study is ACE-TAO. ACE stands for the Adaptive Communication Environment and TAO stands for The ACE ORB. Further information on ACE-TAO is as follows:

1) ACE
- An open-source object-oriented (OO) framework
- Targeted for developers of high-performance and real-time communication services and applications
- Implements many core design patterns for concurrent communication software

2) TAO
- An open-source, standard-based CORBA middleware framework
- A real-time high-performance ORB
- Built by applying the patterns and components in the ACE framework.

TAO services used for developing COBRA-based Monitor and Control Infrastructure Services (explained in next section):

1) **TAO's Naming Service** maps names to object references:
- It implements OMG Naming Service specification
- Clients can use meaningful names for objects (ex., application subsystem names)
- Clients can use different implementation of an interface w/o changing the source code
- Communication between subsystems can be easily handled by this service.

2) **TAO's Real-Time Event Service** provides support for decoupled communications between suppliers and consumers:
- It implements basic OMG Event Service push model, plus features such as event filtering, event correlation, real-time event scheduling, etc.
- Events can be filtered based on the event type and source id.

6. PROTOTYPED APPLICATION & RESULTS

The application prototype completed and demonstrated included portions of the NASA Deep Space Network (DSN) Monitor & Control Infrastructure Service (MCIS) task using CORBA ACE-TAO. Note that MCIS is an existing task, which went from a DCE to socket-based implementation as part of the Network Control Project upgrade efforts, and was chosen for this prototyping study in order have actual operational requirements to gage the COTS OO-component approach results against. MCIS provides monitor data services (ex. publish/subscribe), monitor control services (ex. event notification, directive/response, or configuration change notice), and functional address or naming services between subsystems within the DSN. The M&C services prototyped and demonstrated included only the publish/subscribe and naming services.

In Figure 6, a high-level overview of the prototyped application architecture is given for the publish/subscribe monitor services relative to a multiple instances scenario. Here, the Monitor Data Server (MDS) collects data from various subsystem publishers (producer in CORBA terminology) and distributes this data upon request to various other subsystems (subscribers or consumers in CORBA terminology). A publisher can be a subscriber and vice versa.

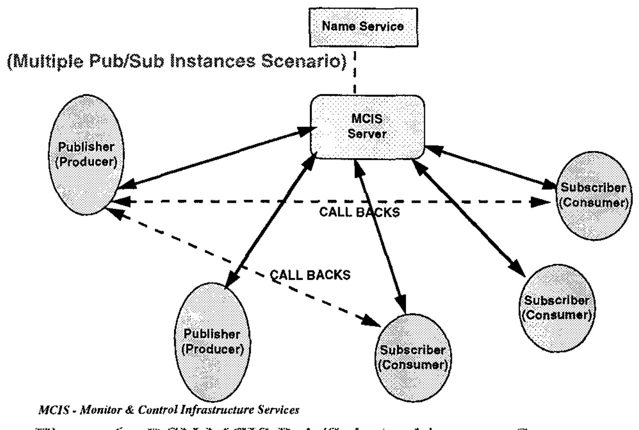


Figure 6 - DSN MCIS Pub/Sub Architecture Summary

In Figure 7, the application structure or configuration using CORBA ACE-TAO is depicted. Basically, the CORBA environment is wrapped to hide it from the developer or user of the MCIS services which provides for a cleaner MCIS API (i.e., existing subsystems use existing API mapped to the new underlying CORBA supporting infrastructure or framework). Thus, the application developer of user subsystem(s) can operator, in this case in a pure C++ and TCP/IP environment, without having to know the rigors of CORBA ACE-TAO. ***The team used ACE-TAO Real-Time Event Service (which should be replaced also by Notification Service) as the backbone to build the MCIS publish/subscribe framework.

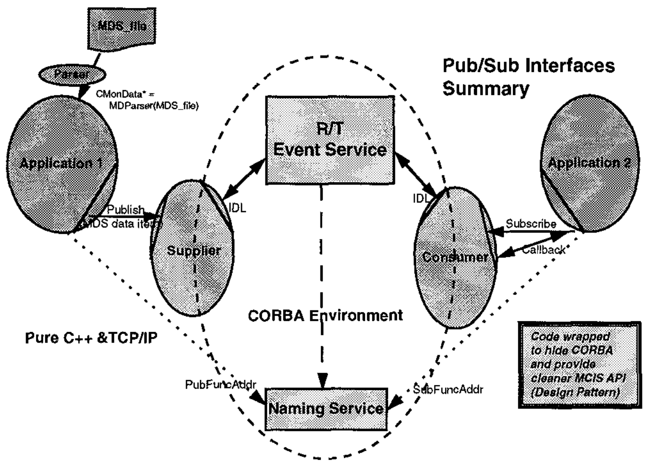


Figure 7 - M&C Pub/Sub Common Services using CORBA-based ACE-TAO

In Figure 8, the COTS versus custom implementation view of the development is shown for the publish/subscribe services (which is only a small subset of the complete MCIS functions). Estimated calculations, which will vary depending on project/coding standards, show that custom development was less than 1% of the total lines of code making up these services across server, consumer, and supplier elements using CORBA ACE-TAO as the underlying service infrastructure. [Note on code comparisons: Team developed = 1541 loc (to use with ACE/TAO), ACE/TAO loaded package = 391907 loc (can be stripped to reduce), JPL custom MCIS socket version = 34109 (no CORBA infrastructure).]

In Figure 9, the CORBA ACE-TAO MCIS application test configuration utilizes a sample subset of typical DSN subsystems which use the underlying Publish/Subscribe services being prototyped. Basically, the combination of Naming and Real-Time Event Services are configured to operate by providing the equivalent of Publish/Subscribe MCIS services between the DSN subsystems [such as Command Control Processor (CCP) and the Ground Comm Facility (GCF) Monitor Processor (GMP)] and execution or data marshaling is carried-out via the Network Monitor & Control (NMC) subsystem.

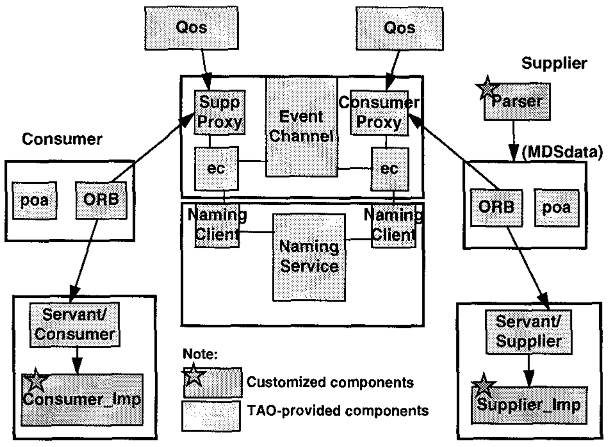
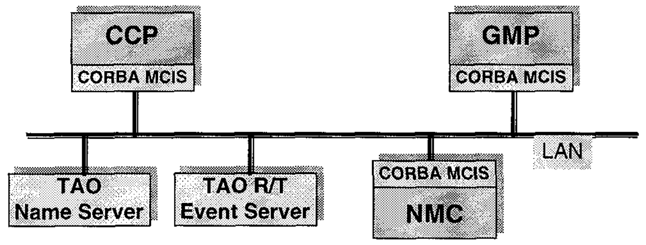


Figure 8 - CORBA Components in MCIS Implementation



NMC - Network Monitor and Controller Subsystem
GMP - GCF Monitor Processor Subsystem
CCP - Command Control Processor Subsystem
▨ - TAO services that run on the same or different hosts
▨ - applications that run on the same or different hosts

Figure 9 - CORBA MCIS Prototype Test Configuration

A summary of achievements of the COTS-based OO-component Monitor & Control Infrastructure Service implementation are as follows:

- Used OO-component approach to show flexibility in system development
- Showed CORBA information hiding
- Showed portable APIs across infrastructure to support changes
- Demonstrated COTS swapping (DCE, ACE/TAO)
- Gained experience in migrating comm. infrastructure from DCE MCIS to Socket MCIS and now to CORBA MCIS
- Found limitations of ACE/TAO real-time Event Service (some of which have been recently solved in the Notification Service).

7. LESSONS LEARNED

The challenges of developing efficient, robust, extensible concurrent applications are difficult. Distributed computing addresses complex topics which are not easily solved. These complex areas are less problematic or not relevant for non-concurrent, stand alone applications.

Some of the lessons learned from using CORBA and the ACE-TAO implementation of CORBA are briefly discussed here in terms of a summary of pros and cons. Table 1 gives a brief listing of key lessons learned using the CORBA standard and Table 2 gives a brief list as a result of using the ACE-TAO implementation of CORBA.

Table 1 - Lessons Learned (Pros & Cons of CORBA)

PROS of CORBA	CONS of CORBA
CORBA is a platform independent distributed computing solution. It is a standard - http://www.omg.org	There is a very steep learning curve with CORBA. It requires strong understanding C++ and IDL.
CORBA IDL (Interface Definition Language) defines a language independent specification, as well as language mappings for most of the major languages.	Requires understanding distributed computing paradigms; e.g. pub/sub remote procedure calls, etc.
CORBA provides many tools to solve very many disparate system engineering problems.	• CORBA assumes you are familiar with OOP principles such as polymorphism, abstraction, etc.
There are many ORB implementations - Compared to DCE, where there has only been one (Transarc) vendor.	• Not all ORBs are created equal! - Many ORBs do not support the latest CORBA versions (versions 2.3 - 3.0).
CORBA allows you 100% leverage of your legacy systems, so old code can be encapsulated under IDL interfaces.	• Commercial ORBs tend to be expensive > $10,000.
CORBA allows integration of other Component Models such as EJB and COM.	• Steep learning curve.
CORBA provides many standard services out of the box. Traditionally at JPL, tools and services have been developed to support distributed computing applications.	•Configuration of systems may be challenging (language compiler versions, OS versions, exception handling, etc).
CORBA rapidly accelerates the development time for complex applications. E.g. this demo took less than 2 months to code.	• Configuration of the ORB is non-trivial (specifically configuring services), due to non-standard implementation across COTS.
CORBA provides an Interoperable Naming Service (INS) which automates the object name mapping.	• Requires an understanding of dynamic memory issues and dynamic types (_var types, type any).
CORBA provides an Event or Notification Service - Note under DCE, JPL created	• Full implementation of services is not guaranteed in all COTS.

Table 2 - Lessons Learned (Pros & Cons of ACE-TAO)

Pros of ACE-TAO	Cons of ACE-TAO
The ACE-TAO ORB is open source so debugging is simplified by the scrutiny of many users.	System requirements are large and involve sys-admin assistance (compilers, OS upgrades, etc).
ACE-TAO provides very good debug-level messages, so developers can easily identify problems.	Large disk footprint > 2 Gigs !! It forced team to obtain a new disk drive.
ACE-TAO implements many of the CORBA Services and even extends them (real times services, etc).	ACE-TAO make-file rules to buildACE-TAO components can be complicated.
There is a large user base in the scientific and business community, and lots of user feedback, and a very active mailing list.	Though ACE-TAO supports many CORBA services, it may not fully implement them, or may provide a non-standard service.
Lots of direct support from the DOC team at UCI and Washington University.	Limited documentation - The documentation assumes knowledge of CORBA.
In developing our applications, the performance was comparable to applications developed using DCE (in our small testbed). I.e., no noticeable performance degradation.	The Event Service is light-weight. Recently received full Notification implementation is a plus.
The Name Service is easily configurable and provides a novel way for clients and	Both the Event and Name Service may leave behind ghosts when killed (i.e., dead processes showing up as active), which can interfere with a new Service.
The Event Service is also very easy to configure.	

JPL may be less prepared to create and maintain in-house infrastructure service solutions for distributed computing across projects and/or organizations. Developing in-house solutions is the "wrong-way" to go for many reasons (developers leave, documentation, debugging, etc.).

The scope of distributed computing is too large for JPL to attempt to try and create any solutions. JPL's focus is on developing and executing spacecraft missions. Point - shouldn't try to create what you can already buy/download!

The correct way to approach distributed computing within NASA/JPL is by adopting open standards. Open standards like CORBA address all areas of complexity that arise in interconnected systems. It is this complexity that forces the need for standards. Not adopting open standards leads to continuous reinvention.

Some sources of complexity created when ad-hoc systems are (re) invented in-house include:

- System latency problems
- Questionable system reliability
- How to synchronize systems, methods
- Deadlocks and race conditions
- Lack of software portability
- Lack of software scalability
- Poor reentrant, type-safe and extensible system call interfaces
- Inadequate debugging and lack of distributed program analysis tools
- Mysterious errors … "gremlins", from errors that are not well understood.

8. ACKNOWLEDGMENTS

Special acknowledgment is given for the significant technical contributions made by the software engineering and prototype development team. This team was composed of Judy Yin, Amalaye Oyake, and Chialan Hwang who successfully prototyped and demonstrated real-time M&C publish/subscribe and naming services within the Deep Space Network (DSN) using this COTS-based OO-component approach. Also, thanks is extended to Chaw Hung for providing system engineering and DSN domain knowledge support. All individuals named, including paper author whose role was task manager and system engineer, composed the ReUse/OO-Component Team.

The research described in this paper was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

9. CONCLUSIONS

Industry and gradually NASA/JPL are moving towards distributed computing infrastructure standards and CORBA-based is currently the most common. With COTS-based infrastructure services (or middleware), system growth and migration can be achieved more effectively and can easily incorporate other services as needed/available (ex., CORBA security services). The COTS-based OO-component approach to software system development promotes reusablilty and allows adjustments or changes along with technology trends.

10. References

The following are references use to support this work:

[1] Doug Schmidt, "Patterns and Frameworks for Concurrent Network Programming with ACE and C++", URL at http://www.eng.uci.edu/~schmidt.

[2] Laverne Hall, C. Hung, & I. Lin, "NASA JPL Distributed Systems Technology (DST) Object-Oriented Component Approach for Software Inter-operability and Reuse", SCI/ISAS'99 - RACDIS'99, Orlando FL.

[3] Object Management Group (OMG) document, The Common Object Request Broker: Architecture and Specification, Vol. I & II, revised February 1998.

[4] OMG document, CORBAServices: Common Object Service Specification, Vol. I & II, updated December 1998.

[5] OMG document, CORBAFacilities: Architecture and Specification, updated 1998.

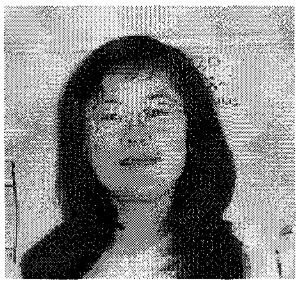
[6] Doug Schmidt, "TAO Developer's Guide", Distributed Object Computing Group at Washington Univ. in St. Louis MI, Object Computing Inc. (ociweb.com), 1999.

[7] Doug Schmidt, Nanbor Wang, and David Levine, "Optimizing the CORBA Component Model for High-Performance and Real-time Applications", Middleware 2000 Conference, New York, April 2000.

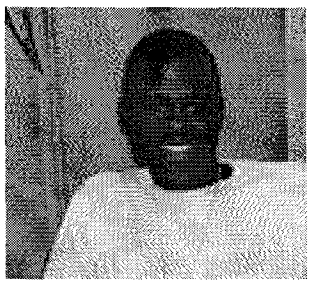
[8] Stanley B. Lippman & Jose'e Lajoie, C++ Primer - 3rd Ed., Addison-Wesley, 1998.

Laverne Hall *has maintained a career in systems engineering and is currently the Technical Group Supervisor for the Distributed Computing & Systems Engineering Group within the Mission Software Systems Section at the Jet Propulsion Laboratory (JPL). She serves on the executive board of the National Council of Black Engineers and Scientists (NCBES) and is an instructor of mathematics at L.A. Southwest College. She has written, presented, and published several papers in the areas of algorithm development and system architecture modeling & simulation (both for satellite-based onboard computing and ground-based processing systems), object-oriented component-based approach to software reuse, and is currently working to promote using this technology approach to software systems development at JPL. Laverne received a BS in Applied Mathematics with a minor in Computer Science from Tuskegee University and a MS in Computer Engineering from the University of California at Los Angeles (UCLA).*

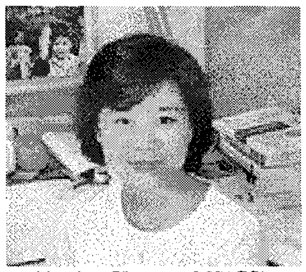
ReUse/OO-Component Team-Members



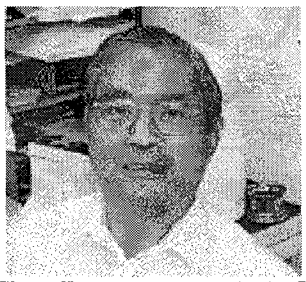
Shyuan-Ju Yin, MS(CS)



Amalaye Oyake, BS(C&SE)



Chialan Hwang, MS(CS)



Chaw-Kwei Hung, PhD(EE&CS)